Drei Komplexitätsresultate zum Mutual-Exclusion-Problem

Prof. Dr. David Sabel

LFE Theoretische Informatik

Letzte Änderung der Folien: 2. November 2020

Übersicht

1. Eine genaue Schranke für den Platzbedarf
2. Ein Resultat zur Laufzeit

Komplexitätsresultate

- Platzbedarf
- Zeitbedarf
- Beachte: Modell ist immer noch: Nur atomare Lese- und Schreibbefehle für den gemeinsamen Speicher.

Eine untere Schranke für den Platzbedarf

Theorem

Jeder Deadlock-freie Mutual-Exclusion Algorithmus für n Prozesse benötigt mindestens n gemeinsam genutzte Speicherplätze.

Eine untere Schranke für den Platzbedarf

Theorem

Jeder Deadlock-freie Mutual-Exclusion Algorithmus für n Prozesse benötigt mindestens n gemeinsam genutzte Speicherplätze.

Bemerkungen zum Beweis (1980 von J.Burns und N.A. Lynch):

- Bei Single-Writer, Multiple-Reader Algorithmen einfach, da jeder Prozess, den Speicher verändern muss, bevor er den kritischen Abschnitt erkennt, sonst wäre wechselseitiger Ausschluss unmöglich (Prozesse müssen erkennen, ob ein anderer Prozess im kritischen Abschnitt ist.)

Eine untere Schranke für den Platzbedarf

Theorem

Jeder Deadlock-freie Mutual-Exclusion Algorithmus für n Prozesse benötigt mindestens n gemeinsam genutzte Speicherplätze.

Bemerkungen zum Beweis (1980 von J.Burns und N.A. Lynch):

- Bei Single-Writer, Multiple-Reader Algorithmen einfach, da jeder Prozess, den Speicher verändern muss, bevor er den kritischen Abschnitt erkennt, sonst wäre wechselseitiger Ausschluss unmöglich (Prozesse müssen erkennen, ob ein anderer Prozess im kritischen Abschnitt ist.)
- Der allgemeine Beweis ist relativ kompliziert. Teile der Argumentation sind, dass mit weniger Speicherplätzen, nicht unterschieden werden kann, ob ein Prozess im restlichen Code oder im Kritischen Abschnitt ist, woraus sich wechselseitiger Ausschluss widerlegen lässt.
- Wir lassen den Beweis weg.

Eine obere Schranke für den Platzbedarf

Theorem

Es gibt einen Deadlock-freien Mutual-Exclusion-Algorithmus für n Prozesse der n gemeinsame Bits verwendet.

Beweis:

- Ein-Bit-Algorithmus
- sowohl J.E.Burns im Jahr 1981 als auch von L.Lamport im Jahr 1986

Idee des Ein-Bit-Algorithmus

Initial: für $i = 1, \ldots n$ want[i]= False,

Programm des i. Prozesses

```
loop forever
(1)   restlicher Code
(2)   want[i]:= True;
(3)   for local:= 1 to n do
(4)     if local≠ i then await want[local] = False;
(5)   Kritischer Abschnitt
(6)   want[i] = False
end loop
```

Idee des Ein-Bit-Algorithmus

Initial: für $i = 1, \ldots n$ want[i]= False,

Programm des i. Prozesses

```
loop forever
(1)   restlicher Code
(2)   want[i]:= True;
(3)   for local:= 1 to n do
(4)     if local≠ i then await want[local] = False;
(5)   Kritischer Abschnitt
(6)   want[i] = False
end loop
```

- Erfüllt wechselseitigen Ausschluss:
  Der erste Prozess im Kritischen Abschnitt hat want auf True, jeder andere wird dies lesen und stecken bleiben
- Erfüllt nicht die Deadlock-Freiheit
- Algorithmus so falsch!

Der Ein-Bit-Algorithmus

Initial: für $i = 1, \ldots n$ want[i]= False,

Programm des i. Prozesses

```
loop forever
(1)  restlicher Code
(2)  repeat
(3)     want[i]:= True;
(4)     local := 1:
(5)     while (want[i] = True) and (local < i) do
(6)        if want[local] = True then
(7)           want[i] := False;
(8)           await want[local] = False;
(9)        local := local + 1
(10) until want[i] = True;
(11) for local:= i+1 to n do
(12)    await want[local] = False;
(13) Kritischer Abschnitt
(14) want[i] = False
end loop
```

Der Ein-Bit-Algorithmus

Initial: für $i = 1, \ldots n$ want[i]= False,

Programm des i. Prozesses

```
loop forever
(1)  restlicher Code
(2)  repeat
(3)    want[i]:= True;
(4)    local := 1:
(5)    while (want[i] = True) and (local < i) do
(6)      if want[local] = True then
(7)          want[i] := False;
(8)          await want[local] = False;
(9)      local := local + 1
(10) until want[i] = True;
(11) for local:= i+1 to n do
(12)   await want[local] = False;
(13) Kritischer Abschnitt
(14) want[i] = False
end loop
```

Tests für $j = 1 \ldots i-1$

Tests für $j = i+1 \ldots n$

- Idee im Groben wie vorher:
  Teste alle anderen want-Wert auf False, bevor in den KA eingetreten wird.
- Daher: wechselseitiger Ausschluss ist erfüllt.

Der Ein-Bit-Algorithmus (2)

Initial: für $i = 1, \ldots n$ want[i]= False,

Programm des i. Prozesses

```
loop forever
(1)  restlicher Code
(2)  repeat
(3)    want[i]:= True;
(4)    local := 1:
(5)    while (want[i] = True) and (local < i) do
(6)      if want[local] = True then
(7)        want[i] := False;
(8)        await want[local] = False;
(9)      local := local + 1
(10) until want[i] = True;
(11) for local:= i+1 to n do
(12)   await want[local] = False;
(13) Kritischer Abschnitt
(14) want[i] = False
end loop
```

Deadlock-Freiheit:

- Beweis-Idee: Bei Deadlock-Berechnungsfolge kann man schließen:
- Irgendwann alle Prozesse:
  - await in Zeile (8)
  - in der for-Schleife in Zeilen (11)-(12)
  - oder für immer im restlichen Code
- und: mind. ein Prozess in der for-Schleife
- Prozess mit größter Nummer wird for-Schleife durchlaufen

Eigenschaften des Ein-Bit-Algorithmus

- Garantiert wechselseitigen Ausschluss und Deadlock-Freiheit
- Starvation ist möglich
- Nicht symmetrisch: Z.B. Prozess mit Nummer 1 durchläuft die `repeat`-Schleife sofort
- Nicht schnell: Wenn nur ein Prozess in den KA will, muss er alle n-Bits testen
- Aber: Platz-optimal, da nur n-Bits gemeinsamer Speicher

Ein Resultat zur Laufzeit

Theorem (R. Alur und G.Taubenfeld, 1992)

Es gibt keinen (Deadlock-freien) Mutual-Exclusion-Algorithmus für 2 (oder auch n) Prozesse, der eine obere Schranke hat für die Anzahl an Speicherzugriffen (des gemeinsamen Speichers), die ein Prozess ausführen muss, bevor er den kritischen Abschnitt betreten darf.

- D.h. Prozesse müssen beliebig lang "warten", bis sie in den kritischen Abschnitt dürfen
- Es gibt keinen Algorithmus der das verhindern kann
- Achtung: Für dieses Modell (Lese- und Schreiboperationen atomar)!
- Resultat meint alle Fälle, es gibt Unterfälle in denen man eine Schranke angeben kann
- z.B.: Fall, in dem nur ein Prozess in den kritischen Abschnitt will

Beweis

Sei M ein Deadlock-freier Mutual-Exclusion-Algorithmus für 2 Prozesse P_1 und P_2

Berechnungsbaum T_M für M:

- binärer Baum
- Jeder Knoten entspricht Zustand der Ausführung (alle Belegungen)
- Prozessinterne Schritte (Schritte, die nicht auf dem Speicher operieren), sind als ein Knoten dargestellt.
- Wurzel: Erster interessanter Zustand: P_1 und P_2 direkt vor dem Eintritt in Initialsierungscode
- linkes Kind eines Knotens: Nachfolgezustand nach einem Schritt von P_1
- rechtes Kind eines Knotens: Nachfolgezustand nach einem Schritt von P_2
- Blatt: P_1 oder P_2 hat kritischen Abschnitt betreten (dann stoppe)

Beweis(2)

Markierung der Knoten von T_M

- Blatt ist genau mit 1 oder genau mit 2 markiert, jenachdem welches P_i im KA ist
- innerer Knoten ist mit 1, 2 oder (1 und 2) markiert, je nachdem wie seine Kinder markiert sind.

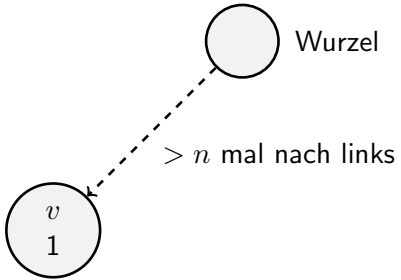
Ähnlichkeit

- Zwei Knoten v, w sind ähnlich bzgl. P_i (geschrieben $v\langle P_i\rangle w$), gdw.
  - Schritte die P_i von der Wurzel zu v macht $=$ Schritte die P_i von der Wurzel zu w macht
  - Gemeinsame Variablen und lokalen Variablen von P_i sind identisch für v und w

Beweis (3)

Theorem ist bewiesen wenn:

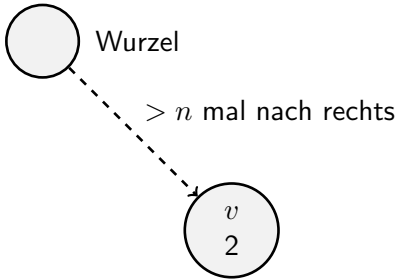
Für jedes $n > 0$ und $i \in \{1, 2\}$:
Es gibt ein Blatt v mit Markierung i, sodass
auf dem Pfad von der Wurzel bis zu v
werden mehr als n Schritte für Prozess P_i ausgeführt



Beweis (3)

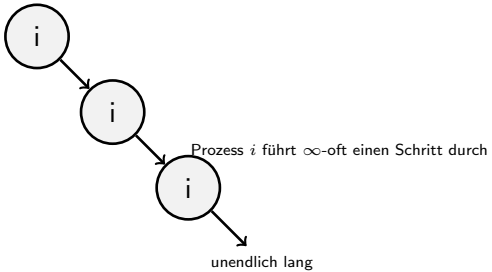
Theorem ist bewiesen wenn:

Für jedes $n > 0$ und $i \in \{1, 2\}$:
Es gibt ein Blatt v mit Markierung i, sodass
auf dem Pfad von der Wurzel bis zu v
werden mehr als n Schritte für Prozess P_i ausgeführt



Beweis (4)

Fall: Es gibt unendlichen langen Pfad in T_M, der unendlich viele Knoten enthält, die alle mit i markiert sind und Prozess P_i führt unendlich viele Schritte auf diesem Pfad aus.



Dann: Für jedes n kann der gesuchte Pfad konstruiert werden.
Deshalb: Annahme A: T_M hat keinen solchen unendlichen Pfad

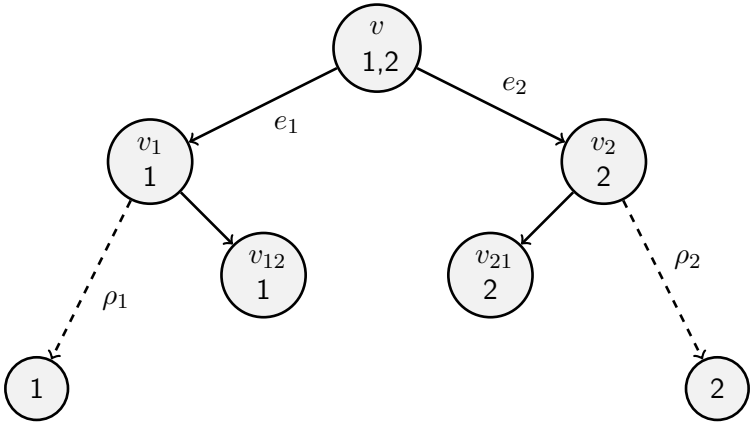
Beweis (5)

Wir zeigen nun: Annahme A führt zum Widerspruch.

Da Algorithmus Deadlock-frei muss gelten (w.g. Annahme A):
Es gibt Knoten v, v_1, v_2 mit

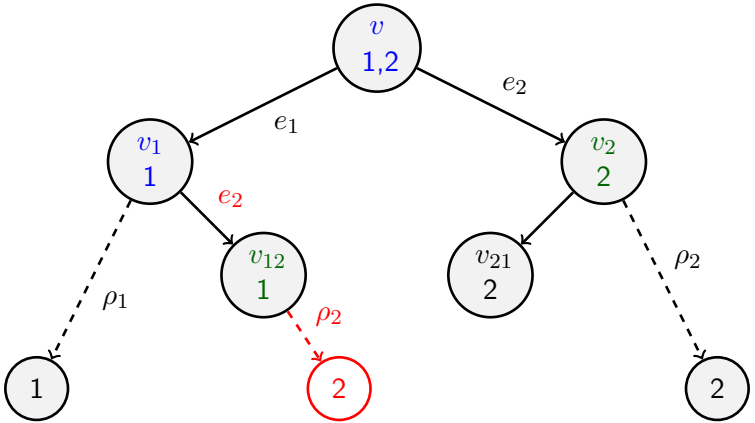
- v ist mit $1, 2$ markiert
- Die beiden Knoten v_1 und v_2 sind jeweils mit genau einer Zahl markiert.

Beweis (6): Fall 1: v_1 mit 1, v_2 mit 2 markiert



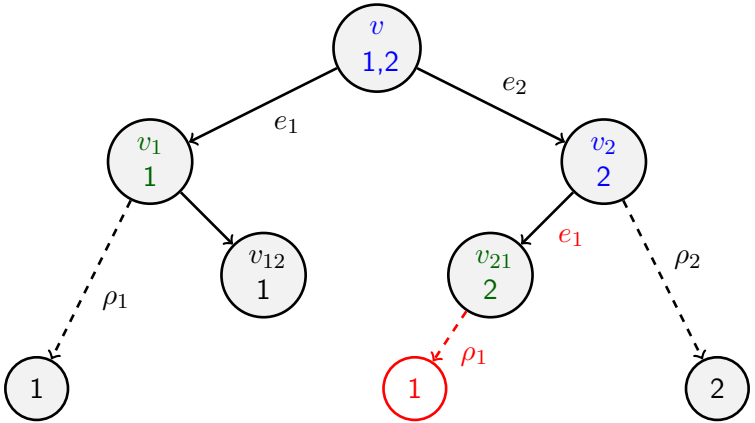
Annahme A $\Longrightarrow$ linkester Pfad endlich lang, rechtester Pfad endlich lang
(Blatt mit 1 bzw. 2 markiert)

Beweis (6): Fall 1: v_1 mit 1, v_2 mit 2 markiert



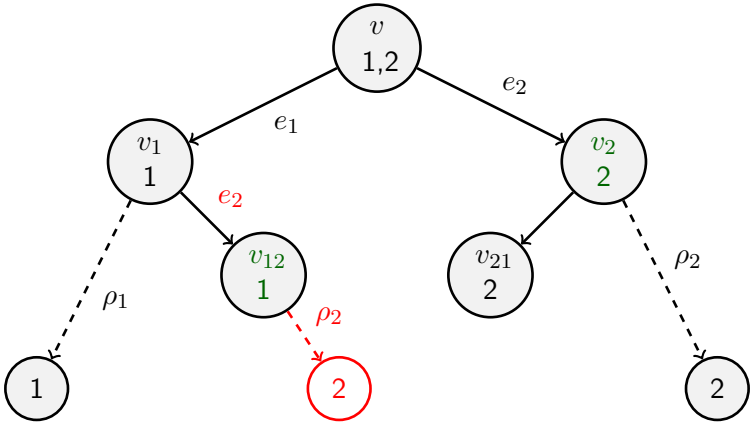
e_1 Lese-Operation: Dann gilt $v\langle P_2\rangle v_1$ und auch $v_2\langle P_2\rangle v_{12}$.
Widerspruch, da ρ_2 auch für v_{12} zu Blatt mit 2 führen muss

Beweis (6): Fall 1: v_1 mit 1, v_2 mit 2 markiert



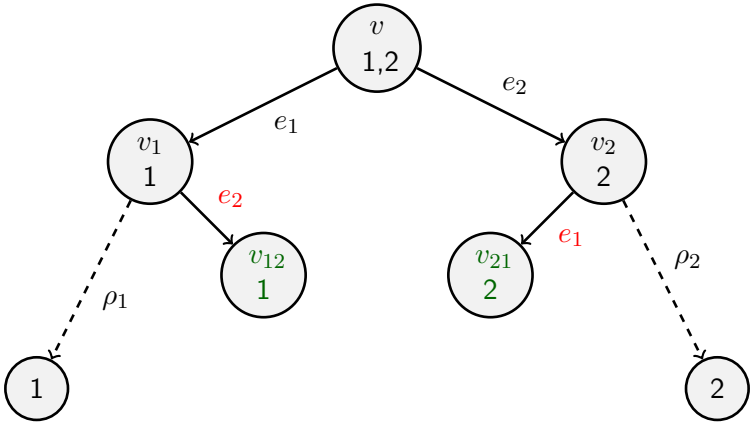
e_2 Lese-Operation: Dann gilt $v\langle P_1\rangle v_2$ und auch $v_1\langle P_1\rangle v_{21}$.
Widerspruch, da ρ_1 auch für v_{21} zu Blatt mit 1 führen muss

Beweis (6): Fall 1: v_1 mit 1, v_2 mit 2 markiert



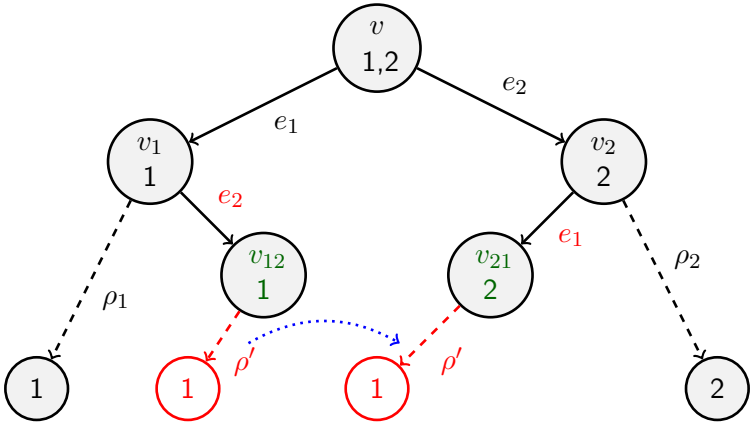
e_1 und e_2 Schreibe-Operation auf gleiche Variablen:
Dann gilt $v_2\langle P_2\rangle v_{12}$ Widerspruch (wie vorher)

Beweis (6): Fall 1: v_1 mit 1, v_2 mit 2 markiert



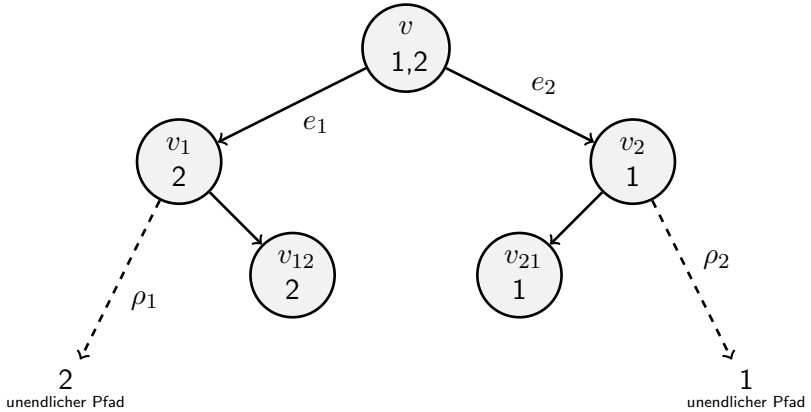
e_1 und e_2 Schreibe-Operation auf verschiedene Variablen:
Dann gilt $v_{12}\langle P_i \rangle v_{21}$ für $i = 1, 2$.

Beweis (6): Fall 1: v_1 mit 1, v_2 mit 2 markiert



e_1 und e_2 Schreibe-Operation auf verschiedene Variablen:
Dann gilt $v_{12}\langle P_i\rangle v_{21}$ für $i = 1, 2$. Widerspruch: Da Markierungen unmöglich

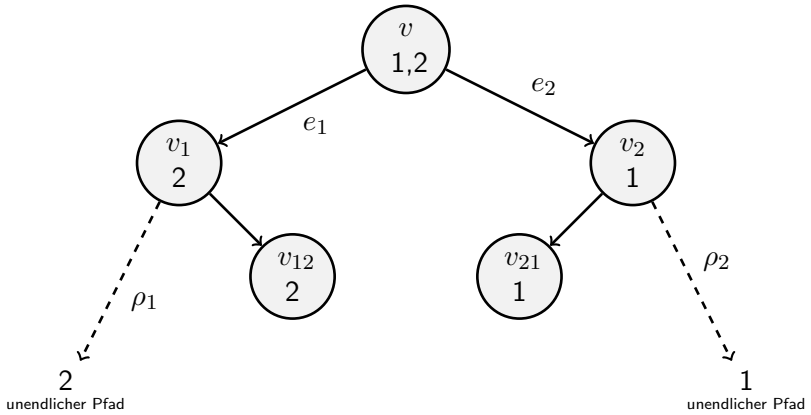
Beweis (7): Fall 2: v_1 mit 2, v_2 mit 1 markiert



P_2 nicht im KA in v $\implies$ P_2 nicht im KA in v_1

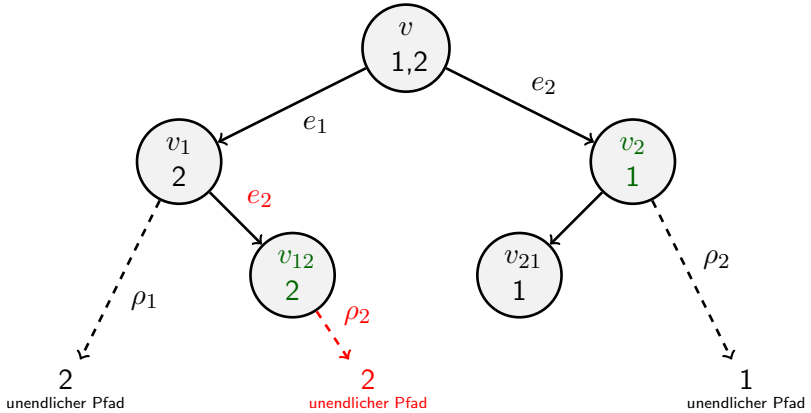
$\implies$ unendlicher Pfad ganz links ab v_1 (da nur P_1 Schritte macht)

Beweis (7): Fall 2: v_1 mit 2, v_2 mit 1 markiert



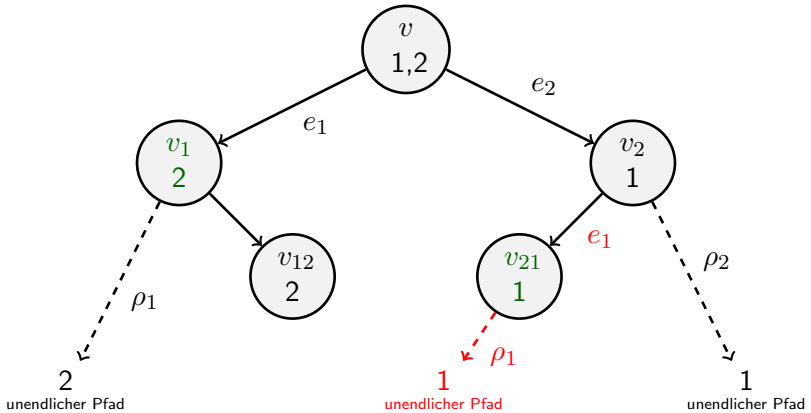
analog: unendlicher Pfad ganz rechts ab v_2

Beweis (7): Fall 2: v_1 mit 2, v_2 mit 1 markiert



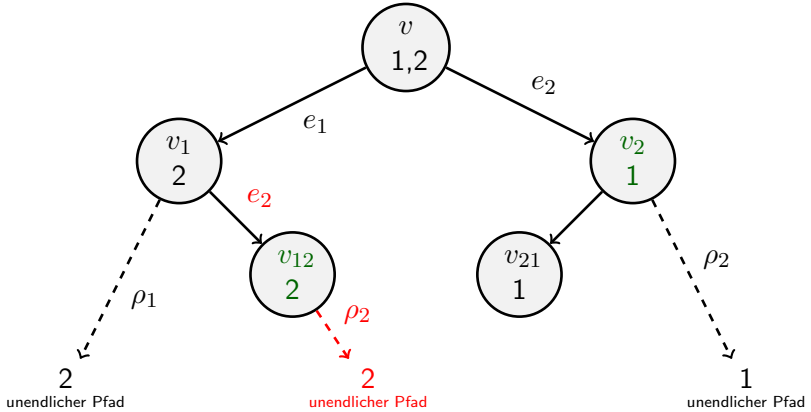
e_1 ist Leseoperation $\Longrightarrow$ $v_2\langle P_2\rangle v_{12}$.
$\Longrightarrow$ ρ_2 auch von v_{12} ausführbar.
$\Longrightarrow$ unendlicher langer Pfad aus P_2-Schritten,
alle Knoten mit 2 markiert. Widerspruch zu Annahme A.

Beweis (7): Fall 2: v_1 mit 2, v_2 mit 1 markiert



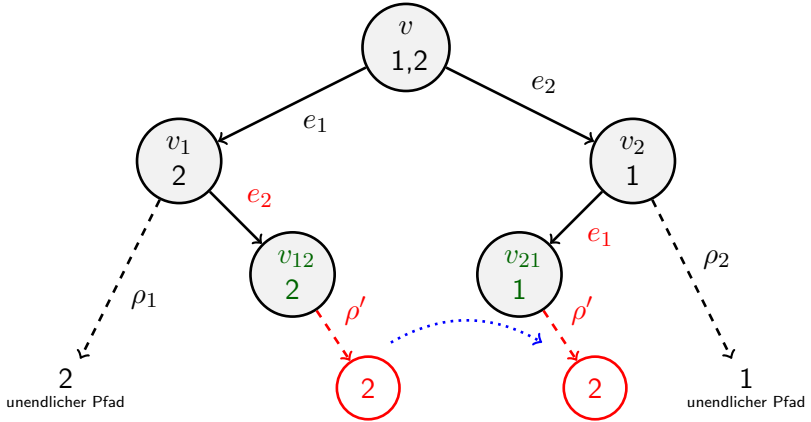
e_2 ist Leseoperation: analog

Beweis (7): Fall 2: v_1 mit 2, v_2 mit 1 markiert



e_1 und e_2 Schreiboperationen in die gleiche Variable. Dann gilt $v_{12}\langle P_2\rangle v_2$.
Widerspruch.

Beweis (7): Fall 2: v_1 mit 2, v_2 mit 1 markiert



e_1 und e_2 Schreiboperationen in verschiedene Variablen. Dann gilt für $i = 1, 2$: $v_{12}\langle P_i \rangle v_{2,1}$. Unmöglich.